VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

STRATEGIE HERNÍCH SYSTÉMŮ

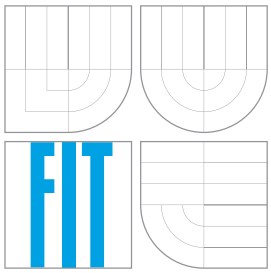
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MAREK ŠVESTKA

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

STRATEGIE HERNÍCH SYSTÉMŮ

STRATEGY FOR GAME SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MAREK ŠVESTKA

VEDOUCÍ PRÁCE
SUPERVISOR

PAVEL ZEMČÍK, prof. Dr. Ing.

BRNO 2015

Abstrakt

Obsahem této práce je reprezentace chování umělé inteligence v počítačových hrách. Cílem bylo najít vhodné metody k její interpretaci. Tato problematika byla vyřešena implementací „map“, což jsou n–rozměrné matice nesoucí informace o průběhu dané hry a dochází k jejich časté aktualizaci. V práci bylo použito několik druhů těchto matic a každá slouží k jinému účelu. Dále byl navrhnut způsob, jak se počítačový protivník může vyvíjet během dané partie hry a vytvořen algoritmus, který mu dává představu o prostoru, ve kterém se nachází. Přínosem práce je způsob, jakým lze „napodobit“ chování skutečného hráče při hraní konkrétní hry. Řešení by také mohlo býti přínosem pro simulátor vojenských bitev ve skutečném světě.

Abstract

This thesis resolves the artificial intelligence representation in computer games. The goal was to find it's suitable interpretation methods. The issue was solved with "map"implementation, which is the n–dimension matrix holding the information about current game state for the concrete player. There were various types of these matrixes used in this thesis. Furthermore was designed a way to evolve computer opponents during the gameplay and created an algorithm, that gives the idea of the space in which it is located. Benefit from this thesis is a solution that makes artificial adversary close to human gameplay behavior. The outcome could also be used in simulators for military battles in real world.

Klíčová slova

umělá inteligence, počítačové hry, strategické hry, rozhodování, plánování, matice

Keywords

artificial intelligence, computer games, strategy games, decision making, planning, matrix

Citace

Marek Švestka: Strategie herních systémů, bakalářská práce, Brno, FIT VUT v Brně, 2015

Strategie herních systémů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana profesora Pavla Zemčíka

.....
Marek Švestka
20. května 2015

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu práce, kterým byl profesor Pavel Zemčík, za odbornou pomoc při práci a přijetí tohoto zadání.

Obsah

Kapitola 1

Úvod

Hry se staly běžnou součástí formy zábavy člověka. Každý si pod pojmem „hra“ představí něco jiného, ale tato práce je však zaměřená na hry počítačové. Počátky počítačových her sahají, až do 60. let minulého století. Mnoho lidí však tehdy nemělo tu možnost vlastnit osobní počítač, „tablet“ nebo „chytrý telefon“. Dnes každého tato zařízení obklopují a s nimi i vyspělé herní systémy. Vývojáři her se však denně setkávají s výzvou vytvořit kvalitní program, z hlediska grafického, dobré hratelnosti i z hlediska kvality umělé inteligence.

Umělá inteligence (angl. *Artificial intelligence*, dále jen `UI`) je dnes stále nejméně probádané téma oboru informačních technologií. „Inteligence“ softwaru se může jevit jakkoliv, například prací stroj může formou SMS zprávy oznámit, že jeho program skončil - jak daleko to až dospěje? Dříve veškeré nástroje vyžadovaly lidský manuální přístup. Dnes se drtivá většina procesů automatizuje a lidé tímto ztrácí v daném úkonu hodnotu. Člověk se díky tomuto směru stává „hloupějším“, nepotřebuje znát detaily procesu, jen potřebuje vědět, k čemu slouží a jak se ovládá. Prostředí kolem nás se naopak stává vyspělejším.

V počítačových hrách jde hlavně o interakci s hráčem. Herní systém prokazuje známky inteligence, pokud reaguje na podněty hráče jistým „rozumným“ směrem. Za dobu existence počítačů však `UI` obrovsky pokročila a přináší v dnešní době svým hráčům opravdovou výzvu. Tato disciplína mě oslovila v raných letech a jako hráč se jí věnuji už osm let, jako vývojář jsem na počátku a budu pokračovat. Chtěl bych tímto „přispět“ do vývojářské komunity počítačových her a rozvinout své schopnosti v tomto oboru.

Hlavním cílem her je poskytnutí zábavy svému hráči. Důležitými faktory počítačové hry je hratelnost, vzhled a dobrý dojem. Hra musí svého hráče zaujmout, aby se k ní pořád vracel. Dříve se za hru na PC zaplatila jednorázová částka. S příchodem a rozvojem internetu vznikají hry, které jsou hratelné zdarma, ale poskytují nadstandardní podporu nebo balíčky nepřístupné běžným hráčům. Také se na nich určitým směrem rozvinul marketing, kdy vydavatelské firmy hráče „tlačí“ do nákupu hry a následně všech rozšíření, které postupně vydávají. Hry ovšem nejsou cílem pouze zábavy, ale podpory výzkumu `UI`.

Tato práce se zabývá herními systémy, reprezentací dat a jejich vhodné implementaci. Předmětem práce bylo navrhnout vhodný způsob reprezentace `UI` a vytvořit aplikaci, modul, který tato data implementuje a tvoří tím `UI` počítačového protivníka. Aplikaci bylo nutno demonstrovat na vhodném příkladě.

V kapitole 2 je obsaženo shrnutí dosavadních postupů v oboru herní inteligence. Jde o část obsahující teorii a zhodnocení dostupných metod. Návrh aplikace se nachází v kapitole 3, kde je popsáno, jaké metody a za jakým účelem jsou použity. Kapitola 4 sděluje postup implementace celého programu a podrobnější popis důležitých řešení a testování.

Kapitola 2

Současný stav technologií

Počítačové hry se dělí podle různých žánrů. Každý z těchto žánrů svému hráči nabídne různou formu zábavy. Jsou hry, kde člověk nemusí moc přemýšlet, je si vědom veškerých aspektů hry a disponuje jednoduchými prostředky. Takové mohou být například akční hry, které jsou komercí nejrozšířenější. Naopak tomu jsou například hry strategické, které vyžadují plnou soustředěnost hráče a kapku jeho kreativity. V tomto stylu her buduje své základny, řeší výzkum, faktory týkající se vojsk a ekonomii. Každá počítačová hra má jiný cíl, nicméně většinou je cílem danou hru vyhrát. [2]

Experimenty různých firem a jejich neoficiální důkazy ukázaly, že člověk velmi pomalu vyvíjí svůj herní styl. Zjistí totiž, že pokud se mu podařilo uspět nějakým směrem, udělá to znovu a jen málokdy zlepší danou situaci. Většina lidí u složitějších her našla svoji strategii a při opakování provedla náhlé změny pouze v případech, kdy došlo k chybám nebo problémům. Pokud počítačový protivník žádným směrem nerozvíjí svůj herní styl(nebo–li strategii), člověk si navykne na tempo a způsob hry. [10]

2.1 Herní systémy

Herní systém může být reprezentován různými způsoby. Většinou jde o systémy s předem definovanou sadou pravidel, které se aplikují na základě definovaných i nedefinovaných událostí ve hře. Tyto události mohou vzniknout na základě času, uživatelského nebo umělého podnětu. V implementaci se mohou nacházet i střípky podoboru strojové učení (angl. *Machine learning*). To se však v počítačových hrách zatím příliš často nevyskytuje a pokud ano, jde o jednoduchou formu a neobsahuje příliš složitou strukturu algoritmů. Čím více faktorů hry a jejího prostředí ovlivňuje hráče i umělé protivníky, tím složitější implementace vyhodnocování a rozhodování těchto protivníků se v něm nachází [8]. Složitost herních systému se rozděluje na tyto typy:

- Systémy založené na pravidlech
- Systémy založené na konečných automatech
- Adaptivní systémy

2.1.1 Systémy založené na pravidlech

Jeden ze základních typů herních systémů. Entity mají přesně definovaná pravidla chování a jejich pořadí, která mohou být podmíněná nebo nepodmíněná. [8]

Příklad č.1: Entita nepodmíněného systému náhodně používá příkazy např. „posun vpřed“ a „otoč doleva“. Má se-li dostat z místa A do místa B tímto způsobem, nebude mít informace o svém okolí a bude pouze přijímat výsledky techto příkazů, jde o čistě náhodné chování.

Systémy s nepodmíněnými pravidly jsou ty nejjednodušší, entity zde „nepřemýšlí“ – nemají informace o okolním prostředí, interagují s ním náhodně. To znamená, že entita využívá příkazy v daném pořadí a nikdy se nezmění. [8]

Příklad č.2: Entita podmíněného systému používá příkazy např. „posun vpřed“ a „otoč doleva“, a to na základě informací získaných ze svého okolí(vyhýbá se překážkám), nebo jiné. Např. pokud zná svou cestu z místa A do místa B, použije tyto příkazy vhodně pro dosažení výsledku.

Systémy s podmíněnými pravidly mohou být široce škálovatelné dle složitosti algoritmů. Entity se chovají způsobem definovaným ve složitých strukturách podmínek a pravidel. Mohou konat akce na základě událostí(např. pokud je před ním překážka, obejde ji). Jde o základní stavební kámen dalších typů systémů. [8]

2.1.2 Systémy založené na konečných automatech

Tento typ je založen na systémech s podmíněnými pravidly. Reprezentace inteligence je zde definovaná stavy, ve kterých se mohou entity nacházet(vždy se však musí nacházet právě v jednom z nich). Zde však nemusí být striktně definován počáteční stav entity. Konečný stav je ten, kdy entita opouští systém a přestává s ním být v interakci. To však také nemusí být vždy pravda, protože v mnohých hrách jsou entity po ukončení svého „života“ stále v interakci se systémem(je možné od nich stále získávat informace nebo ovlivňují chování jiných entit, atd.). [5]

Příklad č.3: Entita je definovaná stavy „klidný“, „v pohybu“ a „ukončený“. Při vstoupení do hry se nachází ve stavu „klidný“, pro posun do cíle je potřeba přejít do stavu „v pohybu“, a při vstupu do cílového pole přechází do stavu „ukončený“ a opouští systém(hra skončí).

Pravidla, kterými se entita řídí po dobu svého života, se mohou lišit v každém ze svých stavů stejně tak, jako způsob rozhodování mezi pravidly. Tento typ je vhodný pro jednodušší hry(např. hra *Pac-Man*) nebo jako součást rozsáhlejšího systému s užitím složitějších metod implementace `UI`. Nepodporují plánování a predikci, ale reprezentují jistou formu „intuitivní logiky“. [8]

2.1.3 Adaptivní systémy

Rozsáhlé programy aplikující principy strojového učení. Systém se přizpůsobuje aktuálním podmínkám hry a vychází ze zdrojů informací, které shromažďuje z předešlých akcí. Vyvíjí své vlastní strategie[1], snaží se předvídat kroky protivníků atd. Principy strojového učení mohou obrovsky zvýšit paměťovou i časovou složitost programu. V důsledku snížení kombinatorických možností chování, které se lze „naučit“, je potřeba vhodně zvolit návrh implementace.[8]

[1] Plán akcí vedoucí k jistému cíli

2.2 Reprezentace chování

Chování programu vždy určuje nějaký vstup a procedury, které se vstupem nějak manipulují za určitým výstupem. Chování umělých protivníků lze jistým způsobem řídit přímo nebo nepřímo. V této podkapitole se setkáme s datovými strukturami, které ovlivňují rozhodování počítače nepřímo, tedy poskytují protivníkovi určité vstupy a on sám rozhodne, jaké chování dále zvolí.

2.2.1 Potenciální pole

Metoda implementující datové struktury, které napomáhají v rozhodování. Entity generují n-dimenzionální pole, které vychází z jejich středového bodu napříč herní mapě. Hodnoty pole se liší v každém políčku nebo vzdálenosti a vycházejí ze středového bodu entity. Počáteční hodnota je dána parametrem a klesá dle dalšího parametru. V závisloti na vzdálenost dosahu totoho pole se dostane až na nulu, kde končí jeho působnost. Potenciální pole(angl. *Potential fields*) mohou přitahovat(kladné hodnoty) nebo odpuzovat(hodnoty nižší než současné). [5] [6].
Druhy potenciálních polí:

- Statické - po dobu celé herní partie se nemění, je konstantní
- Semi-statické - aktualizace neprobíhá často, změnu pole generuje nějaká událost ve hře
- Dynamické - entity generují pole v konstatních intervalech nebo s každým pohybem

2.2.2 Mapa vlivu

Mapy vlivu(angl. *Influence maps*) jsou velmi podobné potenciálním polím. Mohou se dělit na typy podle jejich významu. Jsou to n-rozměrné pole, které nesou informace o daných místech ve světě hry[3]. Slouží k *real-time* rozhodování, ukládání historie nebo k předvídání budoucích událostí. Metoda není vhodná pro malé světy, a to z důvodu náročné implementace. V takovém případě se používá jednoduché hledání cest a kontrola vzdáleností. Je možné využít 2D či 3D reprezentace map, například mřížka (angl. *2D grids*) pro plošinové hry, nebo síť traťových bodů (angl. *Waypoint network*) pro 3D situace. Tato metoda je vhodná pro terén s velkým počtem překážek, ale v případě řídkých oblastí dochází ke zbytečnému zaplnění paměti - řešením je zhrubšit mřížku dat v dané lokaci. [4]

2.3 Podpůrné algoritmy

V moderních hrách se využívá široká škála algoritmů. Jedny ze stále nepoužívanějších a nejrozšířenějších algoritmů jsou metody prohledávání stavového prostoru. Jejich volba je však důležitý úkol, před kterým vždy stojí vývojáři her. Mohou se totiž velmi lišit jak v časové, paměťové složitosti, ale i v nalezeném výsledku. Metody se dělí na tyto typy [12]:

- Slepé metody(angl. *Blind Search Methods*)
- Informované metody(angl. *Informed Search Methods*)
- Metody lokálního prohledávání

2.3.1 Slepé metody

Neobsahují žádné informace o cílových stavech ani o aktuálním stavu, ve kterém se nachází[12]. Mezi tyto metody patří například:

- Metoda slepého prohledávání do šířky(angl. *Breadth First Search - BFS*)
- Metoda stejných cen(angl. *Uniform Cost Search - UCS*)
- Metoda slepého prohledávání do hlouby(angl. *Depth First Search - DFS*)
- Metoda zpětného navracení(angl. *Backtracking*)

2.3.2 Informované metody

Více sofistikované oproti slepým metodám. Mají totiž k dispozici informaci o cílovém stavu a zároveň i formu hodnocení aktuálního stavu. Jsou to metody velmi blízké lidskému uvažování[12]. Mezi uvažované metody patří například:

- Metoda nejlepšího hledání(angl. *Best First Search*)
- Metoda chamtivého hledání(angl. *Greedy Search*)
- Metoda A*

Mezi nejčastější metody k nalezení cest mezi dvěma body v počítačových hrách se používá metoda A*, která splňuje kritéria úplnosti a optimálnosti[2]. Tato meta se nepoužívá však jen v herních systémech, ale má využití také v různých oborech při nalezení cest k cíli. [13]

2.3.3 Metody lokálního prohledávání

Tyto metody mají navíc pár předností oproti informovaným metodám. Jsou použitelné při velmi rozsáhlých stavových prostorech vzhledem k jejim zanedbatelné paměťové náročnosti. Prohledávájí pouze své okolí, optimální stav. Cesta k cíli je bezvýznamná. Využití těchto metod jsem však v této práci nenalezl. [12]

2.3.4 Metoda hledání pohyblivých cílů

Metoda hledání pohyblivých cílů(angl. *Moving Target Search – MTS*) je algoritmus, který řeší problém hledání stanoveného řešení, které se může během výpočtu různě měnit. Jiné algoritmy počítají, že konečný stav se nezmění během jejich výpočtu narozdíl od metody hledání pohyblivých cílů. [7]

Jde o vyspělý algoritmus, který se uplatňuje při sledování pohyblivých entit v herních systémech. Jeho využití však sahá až po navigační systémy *GPS*[3]. Konečný stav je ten, kdy sledovací objekt dosáhne objektu sledovaného. Existují dva typy hledacích algoritmů dle času výpočtu. Prvním je *Off–line* provedení, například A*, který vypočítá celou cestu již před provedením prvního kroku. Narozdíl od *Real–time* algoritmů, které při každé kroku zjišťují novou cestu a pozici cíle, například *RTA – Real-Time-A**. [7].

[2] Jedny z kritérií pro hodnocení těchto metod

[3] *Global Positioning system* je polohovací systém, který pomáhá při určení polohy kteréhokoliv místa na Zemi.

Algoritmus používá matici heuristických[4] hodnot, které zjišťuje na základě každého možného místa cíle. Uchovává heuristickou funkci $h(x, y)$ kde x a y je pár stavů, kdy z x se dostáváme do y. K těmto výpočtům je použita statická výpočetní funkce heuristické hodnoty. Během výpočtu jsou tyto hodnoty aktualizovány podle rozdílů od těch statických, které vrací tato funkce. [7]

2.4 Herní inteligence

`UI` se v počítačových hrách dělí dle úrovně abstrakce[9]:

- nízkoúrovňové akce elementů hry
- situace ve hře
- taktické chování
- strategické chování
- model soupeře

Nízkoúrovňové akce elementů hry představují nejzákladnější posuny nebo změny stavu jednotlivých entit hry. Jde o situaci, kdy má objekt možnost vykonat jednu ze sady definovaných akcí, čímž může být například pohyb vpřed(2.1.1). Situacemi ve hře se rozumí dosažení určité podmínky ve hře a spuštění dané události, která má následovat(například jednotka dorazila na specifické místo, kde bude zničena). Taktické chování znamená, že objekt rozhoduje(nebo „něco“ za něj rozhoduje), co za akci vykonat na základě stanovených podmínek(například jednotka se stáhne z dosahu nepřítele, pokud je např. těžce zraněná). Strategické chování reprezentuje celkové chování umělého protivníka. Veškeré akce nízších úrovní jsou sloučeny a vzniká tímto daná strategie konkrétního subjektu(hráče). Může jí o strategii dlouhodobou nebo krátkodobou[5]. Model soupeře je velmi podobný strategickému chování. Jde o personalizaci daných strategií určitému umělému protivníkovi s pojmenováním. [9]

2.5 Strojové učení v herních systémech

Aplikace strojového učení do počítačových her není zatím v dnešní době velmi rozšířeno, avšak stále přibývá více a více her, které přináší nové metody řešení. Hráči se tak setkávají s vyspělejší, robustnější a lidskému uvažování podobné `UI`. Je také možnost obtížnost protivníků dynamicky měnit. Nicméně vzhledem k úspornějšímu vývoji inteligence je stále potřeba kontrolovat pečlivě veškeré výsledky, které se mohou principem „samoučení“ vyvinout nesprávným směrem(problémy s chováním během herní partie). Opravy těchto chyb mohou být finančně náročné a tímto také zpomalit vývoj projektu. [9]

2.5.1 Vývoj učení protivníků

Strojové učení v počítačových hrách můžeme rozdělit na dva typy:

[4] Heuristika je řešení problémů podle různých přístupů nebo metod.

[5] Dlouhodobá strategie je plán k dosažení cíle hry. Krátkodobá strategie je lokální odluka od dlouhodobé strategie za cílem získání určitých hodnot.

- Off–line - během vývoje aplikace
- On–line - během samotného hraní hry

Přístupy v obou těchto typech učení se dále rozdělují. Mezi první přístup můžeme zařadit způsob učení sledováním lidského protivníka. Umělý hráč tímto kopíruje akce člověka a upravuje dle zvolených parametrů nebo vlastního(pomocí definovaných pravidel) úsudku. Dalším pohledem je učení hráčem, kdy člověk explicitně dává umělému hráči příkazy a ten si je ukládá pro pozdější použití. Mezi další patří také učení ze zkušeností, kdy probíhá několik partií za sebou tak, že počítač sehrává hru střídavě proti lidskému a umělému hráči. Tímto se vyvíjí nové možné strategie pro budoucí použití. [9]

2.5.2 Aplikace ve hře *DEFCON*

DEFCON je hra pro více hráčů s motivy jaderné války na planetě Zemi. Cílem hry je zničit nepřátele a získat tak nejvíce bodů.

Umělý protivník zde využívá konečný automat, který je sekvencí jeho vývoje během hry. Jde tedy o systém založený na konečných automatech(2.1.2). Při dosažení posledního stavu zůstává protivník v daném stavu po zbytek hry. Jde o stavy následující[11]:

- Umístění pěchoty a námořnictva.
- Výzvědné letecké a námořní mise k odhalení pozic nepřítele.
- Útok bombardéry.
- Plná polní a útok na dané lokace nepřítele.
- Konečný stav, námořnictvo útočí na náhodně spatřené nepřátele.

Jeho chování je však podloženo vyspělými mechanismy. Jde o dvě vrstvy, nižší a vyšší, které tvoří `UI` protivníka. Na nižší vrstvě si ukládá důležité události z předešlých her a pomocí nich předvídá chování svého protivníka. Tyto informace se určitým způsobem ukládají do map vlivu(2.2.2) a jsou využívány k synchronizaci útoků, přesunů nebo formací námořnictva. Na té vyšší jde o systém učení použitý vzhledem k dlouhodobým plánům v dané partii hry. [11]

Strojové učení jde zde aplikováno formou metody simulovaného žíhání[6], rozhodovacích stromů a usuzování na základě událostí. Je zde také použit algoritmus pro umístění klíčových prvků hráče(budovy atd.). Obsahuje systém ohodnocení předchozích rozložení základen a na základě tohoto hodnocení si vybírá různá místa pro nové uložení. [11]

K vyhledání vhodného rozložení základny využívá umělý protivník algoritmu simulované žíhání. Vývojáři hry testovali tento algoritmus s různými parametry a po nalezení nejvhodnějšího nastavení zjistili, že ze 150 herních partií byl umělý hráč schopen zvítězit ve 115 případech, což je zaokrouhleno na 77%. [11]

Systém se tedy stáva více adaptivním herním systémem(2.1.3). Jde o iterativní[7] optimizační přístup k učení a zlepšení „inteligence“ počítače a podobá se řešení evolučních algoritmů. [11]

[6] Stochastická metoda jejímž cílem je překonání lokálních extrémů vedoucích k častým neúspěchům metody *Hill-climbing*[12]. Obě metody patří do odvětví metody lokálního prohledávání.

[7] Dosažení potřebných výsledků opakovým testováním dané situace. Výstup každé iterace je zpětnou vazbou na vývoj.

Kapitola 3

Analýza a návrh aplikace

3.1 Analýza a zhodnocení

3.1.1 Potenciální pole a mapy vlivu

Metodu potenciálních není v práci použita. Kombinace jejích variant by však použitelná byla, ale příliš časté aktualizace daných matic by nejspíše vedlo ke zpomalení výpočetního výkonu. Možnost testování obou metod jsem bohužel neměl, a proto jsem raději zvolil metodů map vlivu(2.2.2), které jsou popsány hned v zápetí. Ačkoliv se potenciální pole používají i k hledání cest k cíli, tuto funkčnost zastoupila jiná metoda zmíněná dále.

3.1.2 Herní systémy

V této práci se vyskytuje manipulace se systémy založené na konečných automatech, což je jeden z vyspělejších základních typů systémů. Do jistého směru se však stává i systémem adaptivním, pokud bereme ohled na aktualizaci map vlivu. Plně adaptivní by se však aplikace stala po implementaci strojového učení, kdy by se partie od partie měnila strategie umělého protivníka. Byl by to zajimavý nápad na další pokračování práce v magisterském směru.

3.1.3 Hledání cest

Pro hledání cest k cíli(angl. *Pathfinding*) byla zvolena metoda A* a to z důvodu toho, že je úplná i optimální. Využití této metody bude popsáno dále(3.2). Dalším kritériem pro zvolení byla četnost výskytů této metody v jiných počítačových hrách [5]. Tato metoda používá však mnoho heuristik. Zvolenou heuristikou je $h = f + g$ kde h je zmíněná funkce, f je vzdálenost od cíle a h je cena aktuálního pohybu na další bod. Tato heuristika vždy najde tu nejlepší cestu, přestože jiné heuristické funkce mohou mít menší dopad na časovou a paměťovou náročnost. [1].

3.2 Návrh

Návrh programu ve formě modulu připojitelného k programu hry. Vzhledem k odlišnostem mezi žánry[1] her i v rámci těchto žánrů je skoro nemožné zaručit znovupoužitelnost modulu

[1] Strategické hry (angl. *Strategy games*), akční hry(angl. *Action games*) a jiné.

tak, aby se dal jednoduše připojit k jiné hře. Z tohoto důvodu budou mezi aplikací a technologickým demem různé propoje, které jsou nezbytné pro funkčnost `UI` hry. Tato místa budou popsána a vysvětlena, z jakého důvodu bylo potřeba daný problém vyřešit zvolenou cestou. Jak již bylo zmíněno, v rámci této práce bylo potřeba zvolit vhodný příklad k demonstraci reprezentace chování. Volně dostupné hry(angl. *Open-source games*) na internetu byly však příliš složité, trvalo by nesčetnou dobu pochopit program a poté dokázat do něj implementovat tento modul. Z tohoto důvodu jsem zvolil vlastní cestu a technologické demo zkonstruoval ve vlastní režii. Programovací jazyk pro veškerý zdrojový kód byl zvolen objektově orientovaný jazyk C++. S tímto jazykem se setkáváme často a i já mám s ním už lecos zažité, proto jsem ho zvolil.

3.3 Zvolený příklad: Technologické demo

Žánr hry jsem zvolil mně velmi blízké strategie v reálném čase(angl. *Real-Time Strategy – RTS*) a se kterými mám největší zkušenosti. Použité zdroje[2] byly vyrobeny pro tento účel externě a jsou v demu dostupné. Aplikace běží v herní knihovně(angl. *framework*, a dále jen framework)[3] Cocos2D-x[4], který je volně dostupný. Důvod volby tohoto enginu, který je multiplatformní, jsou bohaté osobní zkušenosti s tímto výrobkem. Dalším parametrem je vzhled dema, který bude ve formě 2D. To znamená, že aplikace nebude náročná na výpočetní výkon grafického adaptéru. Dále následuje velmi stručný popis návrhu technologického dema.

3.3.1 Popis

Program poběží na jediném vláknu procesoru(angl. *single-threaded*) – herní engine neumí vícevláknové vykreslování(angl. *multi-threaded*), protože je abstrakcí grafické knihovny *OpenGL*[5]. Reprezentace herního pole bude ve formě šachovnice, která ma jasný začátek a konec. Jednotky[6] se pohybují přesně po jejích políčkách. Nelze, aby jednotka zůstala stát na pomezí dvou nebo více políček. Pro hráče bude implementováno jednoduché grafické uživatelské rozhraní, které je velmi podobné tomu ve hrách stejného žánru z dnešních dob. Podrobnější popis zacházení s aplikací bude popsán v příručce obsažené v příloze této práce.

Hra bude obsahovat jednoduchou logiku, kterou je potřeba vytvořit vzhledem k ovladatelnosti hráčem. To znamená, že jsem nemohl nechat všechno rozhodování jen na modulu `UI`. Aplikace řeší vysílání projektilů z jednotek na jiné(vzdálenost, rychlost apod.), přesuny jednotek(velmi jednoduché rozhodování) a také obsahuje ukončující podmínku hry(tzv. cíl hry). Pro tvorbu dalších objektů je nutno vlastnit určité hodnoty měny hry(finance), které se generují automaticky a rovnoceně pro všechny hráče. Za tuto měnu je možnost nakupovat arzenál vozidel nebo budov. Hráč vlastní na začátku hry několik vozidel a hlavní budovu, která po jejím zničení ukončuje hru ve prospěch toho hráče, který ji zničil.

Výčet elementů technologického dema:

- Základna - zázemí a opevnění hráče, na počátku hry je opevnění ve formě hlavní budovy a hrstky vozidel kolem ní
  - Hlavní budova – hlavní stavební element a generátor zdrojů(herní měny)

[2] U her se používá termín zdroj(angl. *Resource*) pro obrázky, zvuky, animace apod.
[3] Knihovna tvořící abstraktní vrstvu mezi komponenty počítače a programem.
[4] http://www.cocos2d-x.org/
[5] *Open Graphics Library* je knihovna pro tvorbu počítačové grafiky, je standardizována a je volně šiřitelná
[6] Pohybující se objekty po ploše hry, které jsou ovladatelné a jsou nástrojem k dosažení cíle hry

  - Malá továrna – výroba vozidel nižší třídy
  - Velká továrna – konstrukce vozidel vyšší třídy
  - Věž s dělem – základní obranná konstrukce, průměrné obranné schopnosti
  - Raketová věž – pokročilá obrana, výtečné obranné schopnosti
- Jednotky – pouze forma vozidel(demo)
  - Výzvědný tank – základní jednotka hry, nejslabší článek vojska
  - Kulometný tank – další základní jednotka, síla v hojném počtu
  - Těžký tank – hlavní úderná síla, dobré bojové schopnosti
  - Raketový tank – podpora ze zálohy, velmi silná jednotka
- Terén – dělí se na vodu a travnatou plochu, voda není jednotkám přístupná, ani konstrukci budov
- Finance - klíčový element k produkci, generován hlavní budovou

V poslední řadě je nutno uvést, že budou použity konfigurační soubory pro inicializaci a logiku hry. Tyto soubory se budou nacházet v daném podadresáři hry a budou analyzovány za běhu programu. Pro reprezentaci dat v těchto souborech jsem zvolil serializační[7] jazyk JSON[8], se kterým mám také bohaté zkušenosti. Soubory se budou členit na:

- konfigurační soubory konkrétních partií – obsahují rozmístění objektů na herní desce, parametry hráčů a globální parametry dané herní partie
- parametry jednotek – definice hodnot každé použité jednotky ve hře(s těmito hodnotami dále pracuje i modul `UI`)

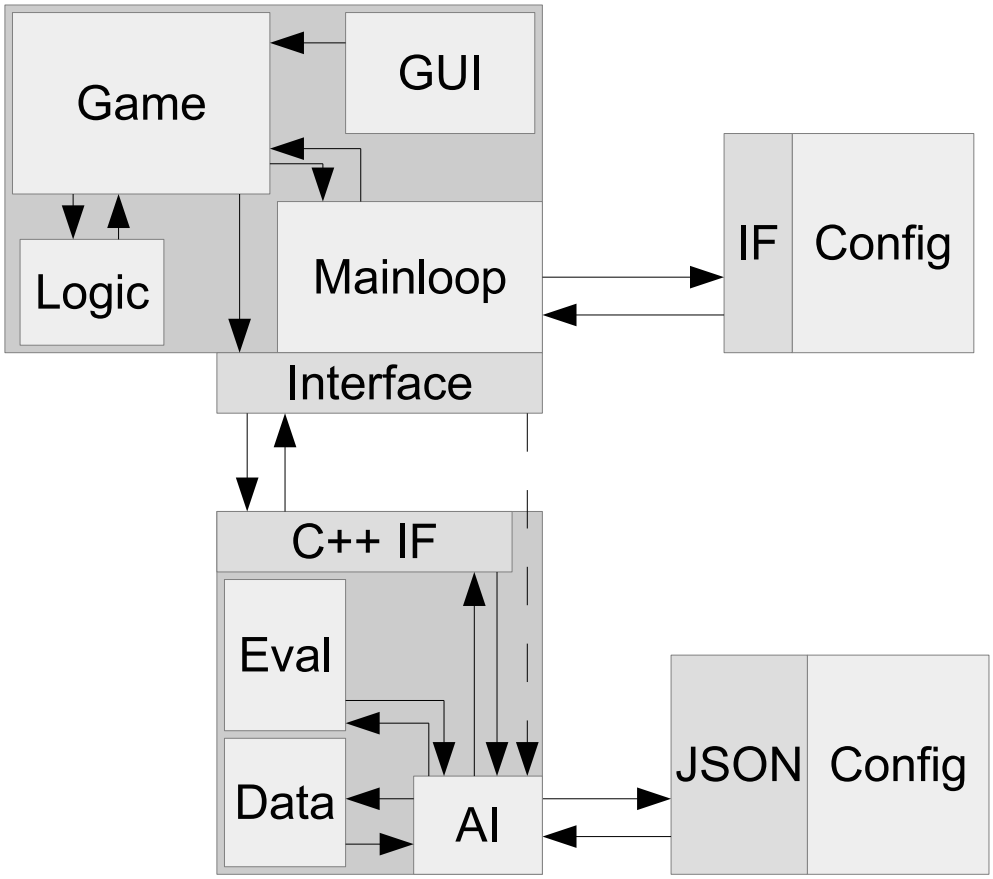
3.4 Architektura modulu

Modul `UI` zastiňuje rozhraní, které slouží ke komunikaci mezi hrou a modulem. Na obrázku 3.1 si lze povšimnout závislostí jednotlivých komponent. *AI* je jádro modulu a využívá *Data*(struktury s datama) a *Eval*(výpočetní jednotka) ke svému fungování. Je také závislé na externí konfiguraci, které je zpracováno rozhraním skrze jazyk *JSON*. Hra má v ojedinělých případech možnost(při speciální implementaci) zeptat se přímo modulu a obejít tím rozhraní.

Na základě požadavků, které vzešly z analýzy, byly navrhnuty následující třídy:

- Interface - třída určená jako komunikační vrstva, kdy hra informuje modul o jistých událostech, které ve hře vznikly
- Computer - hlavní forma, která řídí nejzákladnější záležitosti umělého protivníka, např. plánování strategií
- Commander - modul řízení lokálních akcí jednotek, snaží se dosáhnout herního cíle, aktivně se dotazuje vyhodnocovacích modulů

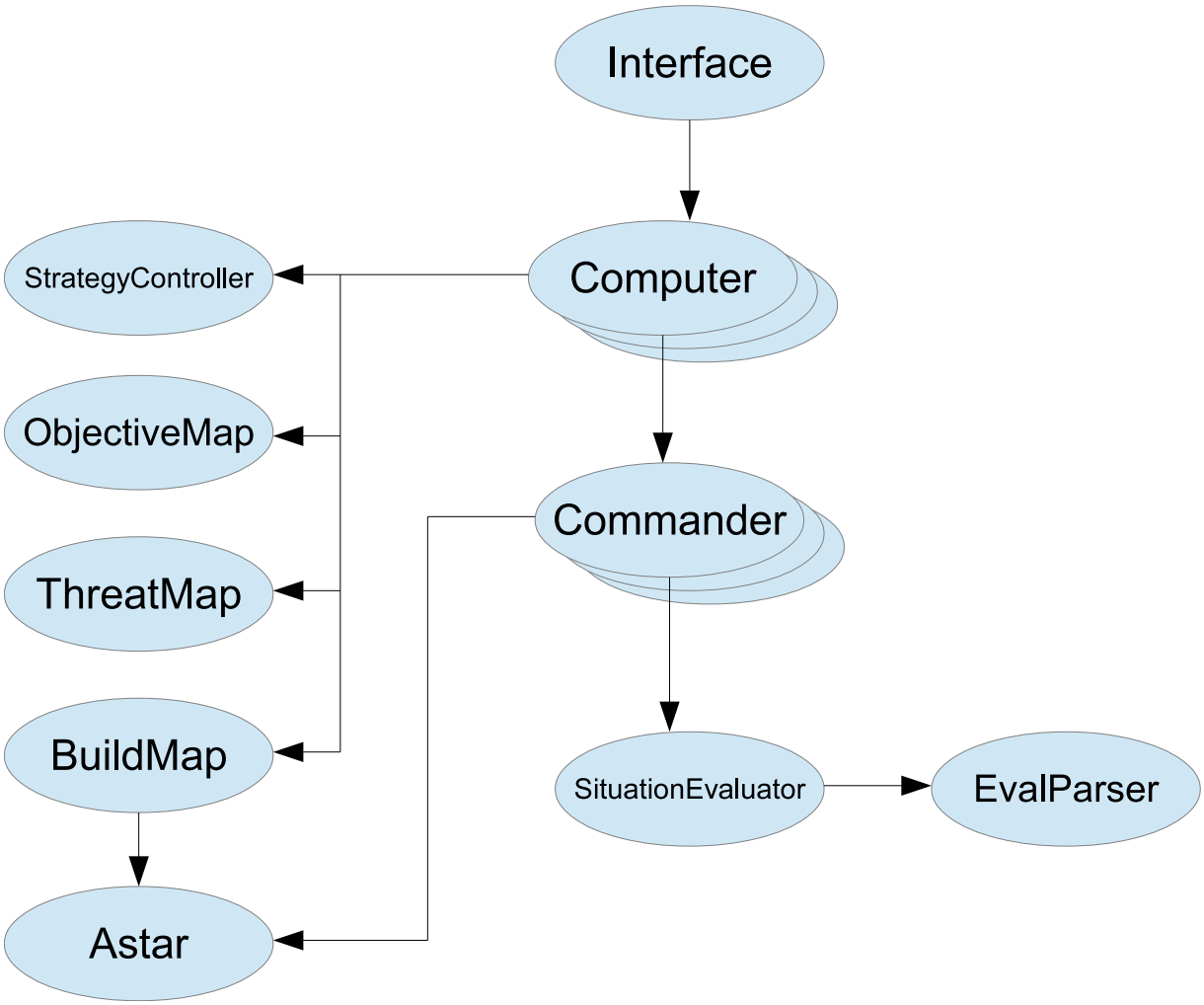
[7] Jazyk pro serializaci dat – serializace je převod objektu do jeho jednorozměrné podoby
[8] JavaScript Object Notation



Obrázek 3.1: Schéma závislostí částí aplikace

- StrategyController - tato třída analyzuje soubory se strategiemi a ukládá je do struktur v programu, jde o analýzu přímých pravidel pro třídu Computer
- EvalParser - zpracovává soubor v JSONu s ohodnocenim jednotlivých parametrů jednotek hry
- SituationEvaluator - vyhodnocovací modul, ohodnocuje atributy jednotky na vstupu a vrací její spočítanou hodnotu, podporuje okamžité rozhodování
- BuildMap - reprezentuje datovou strukturu, která slouží k nepřímému rozhodování při stavění budov na klíčové pozice(viz. 3.5.3)
- ObjectiveMap - matice, která nese data o lokacích spatřených budov nepřítele, slouží k nepřímému rozhodování při plánování útoků(viz. 3.5.1)
- ThreatMap - další z reprezentačních tříd, která slouží pro záznam bitev po herní desce – ovlivňuje pohyb vojsk umělého protivníka(viz. 3.5.2)
- Astar - implementace algoritmu A*

`UI` pracuje v časových intervalech, které spouští jádro hry. Každý tik kdy je spuštěna se hodnotí stav hry, kontroluje se dlouhodobý plán a podle něj vykonává další akce. Na obrázku 3.2 se nachází schéma závislostí tříd modulu.



Obrázek 3.2: Schéma závislostí tříd modulu `UI`

3.5 Definice dat

Pro reprezentaci dat, které sbírá daný umělý protivník, byly zvoleny následující datové struktury:

- Mapa cílů
- Mapa ohrožení
- Konstrukční mapa

Jednotlivé strukury jsou reprezentovány třídami v jazyce C++(3.4). Skrze jejich rozhraní lze číst data z map a aktualizovat dle vzniklých událostí. Jsou to objekty implementující podtstu a logiku mapy vlivu(2.2.2). Jak již bylo zmíněno, jsou n-rozměrné. V této práci se však setkáme hlavně s jejich 2D formou maticí.

Herní deska technologického dema je podobná šachovnici ze hry šachy, tímto se nám selekce určitých míst zjednodušuje. Pokud by se jednotky mohly pohybovat po desce jakkoliv(nebyly omezeny minimální vzdáleností pohybu – 1 políčko), události na daných místech

by se procentuálně rozdělily mezi ty buňky matice, mezi kterýma se událost zrovna nachází. Časová složitost prohledávání okolí entity je tedy nejvýše kvadratická.

U 3D situací přibývá další dimenze značící výškovou polohu. Uvažujme, že se entita nachází uvnitř dvoupatrového domu v přízemí. Při nalezení nepřítele v těchto prostorách se uloží záznam do mapy ohrožení tak, že hodnota značící výšku bude nula(nebo taková hodnota, která reprezentuje pevnou zem v daném světě). Potom tedy při plánování obsazení patra prvního, bude `UI` uvažovat cestu přízemím a velmi pravděpodobně očekávat střet s nepřítelem, pokud není jiné cesty.

Vývojáři tedy volí hustotu matice sami. Jedna buňka matice může být velikosti jedna jednotka vzdálenosti hry(např. 1m) nebo více. Přílišnému „zahušťování“ těchto struktur může docházet k většímu náporu na výpočetní zdroje. Možností jak optimalizovat příliš velké světy, by bylo možno například kompromis velikostí buněk tak, že při detailnějších situacích nebude rozhodování nijak zvlášť rozvráceno(např. pokud je cesta velmi úzká, ale je znázorněna jako nebezpečná, protože došlo k překryvu od nějaké vedlejší akce – za zdí blízkého domu). Další možností by mohlo být vytvoření hybridního přístupu k těmto strukturám, a to tak, že na velké plochy bude použita mapa menší hustoty a na detailní akce mapa větší hustoty buněk. Tímto vznikne více map soustředěných přes herní desku a přístup k nim je potřeba implementovat přes univerzální rozhraní.

3.5.1 Mapa cílů

Hráči vlastní budovy k tomu, aby mohly rozvíjet své armády a stavět nové budovy. Jelikož jsou budovy nehybné, umělý protivník si ukládá pozice nepřátelských budov do mapy cílů, která značí, kam soustředit příští útok. Důležitost daného cíle bude označena celočíselnou hodnotou. Bude to tedy matice čísel, která se bude často aktualizovat. Hlavní cíl hry bude označen nejvyšším číslem, kterým je v technologickém demu budova s radarem.

3.5.2 Mapa ohrožení

Podle dřívějšího nastínění tohoto objektu, jde o strukturu, která ovlivňuje `UI` při přesunech po herní desce. Uvažujme, že vlastníme skupinu vozidel a setkáme se s nepřítelem v poli. Podle dohledu našich vojsk zjistíme, jakému problému zrovna čelíme. Vypočítáme hodnotu nepřátelské armády(4.2.1) a zjistíme, zda máme nějakou šanci uspět. Zde mohou nastat 3 situace:

- Nepřátelské vojsko bylo zničeno – nebude uveden žádný záznam do mapy ohrožení, protože je hrozba zažehnána.
- Nemáme šanci uspět – stáhneme vojsko zpět do bezpečné zóny, při ztrátě dohledu na určité nepřátelské vozidlo aktualizujeme mapu ohrožení s hodnotou dané jednotky(4.2.1) na místě, kde byla entita naposled zahlédnuta
- Padneme v boji – zbylá nepřátelská vojska uložíme do mapy uhrožení

Později při plánování dalších akcí budeme hledět na místa v mapě, kde ke střetu došlo. V těchto místech očekáváme, že je možnost dalšího střetu a vypravíme se do této lokace minimálně s hodnotou vojska takovou, která je součtem dílčích míst v dané lokalitě. Další rozumnou možností by bylo dané místo obejít nebo vyslat zvědy v první linii.

Mapy ohrožení tedy slouží k uchování posledních bojů. Udávají představu o tom, kudy vézt protiútoky a na která místa si dávat pozor. Aby nedošlo k uplnému zaplnění mapy

ohrožení, je nastaven parametr, který každým tikem přiřazeným `UI` snižuje všechny hodnoty v této struktuře. V rámci optimalizace, snížení hodnot by se dalo parametrizovat tak, že by např. ke snížení došlo každý pátý tik o pětkrát danou hodnotu.

3.5.3 Konstrukční mapa

Slouží k budování základny. Dává představu o tom, kde umístit obranné věže a kam postavit produkční budovy, které se nemohou bránit. V mnohých hrách minulého desetiletí se protivníci nijak zvlášť nezabývali kličovými pozicemi budov. Šlo tak vidět například ve hře *Red Alert 2*[9], kde ke konstruování docházelo spíše náhodně a docházelo k různým problémům v neobvyklém terénu. V této práci jsem se však zaměřil na tuto problematiku a vyvinul vlastní algoritmus hledání příchodových cest do základny. Výpočet probíhá na začátku hry a výsledkem je opět dvourozměrná matice celých čísel taková, kde hodnoty blízko nule znamenají nejmenší pravděpodobnost příchodu nepřátel. Vyšší čísla znamenají pravděpodobné napadení v daném místě a zde bude `UI` umísťovat obranné věže. Bližší seznámení s algoritmem v kapitole implementace(4). [11] [3].

[9] RTS hra vydána roku 2000 v tématu studené války firmou *Electronic Arts*

Kapitola 4

Implementace

Celá implementace projektu se nachází ve složce *Classes/*, kde se dále dělí na technologické demo a bakalářskou práci. V této složce už nalezneme demo s podsložkami *src/* a *include/* a modul v adresáři *AI/*. Nelze si nevšimnout knihovny *cJSON*[1], která je použita pro zpracování konfiguračních souborů. Tato knihovna je velmi robustní a jednoduchá, z tohoto důvodu nemohla chybět v jádru programu. Dále následuje nejdříve obecnější popis implementace a později zacházení do mírných detailů nejdůležitějších záležitostí práce.

4.1 Technologické demo

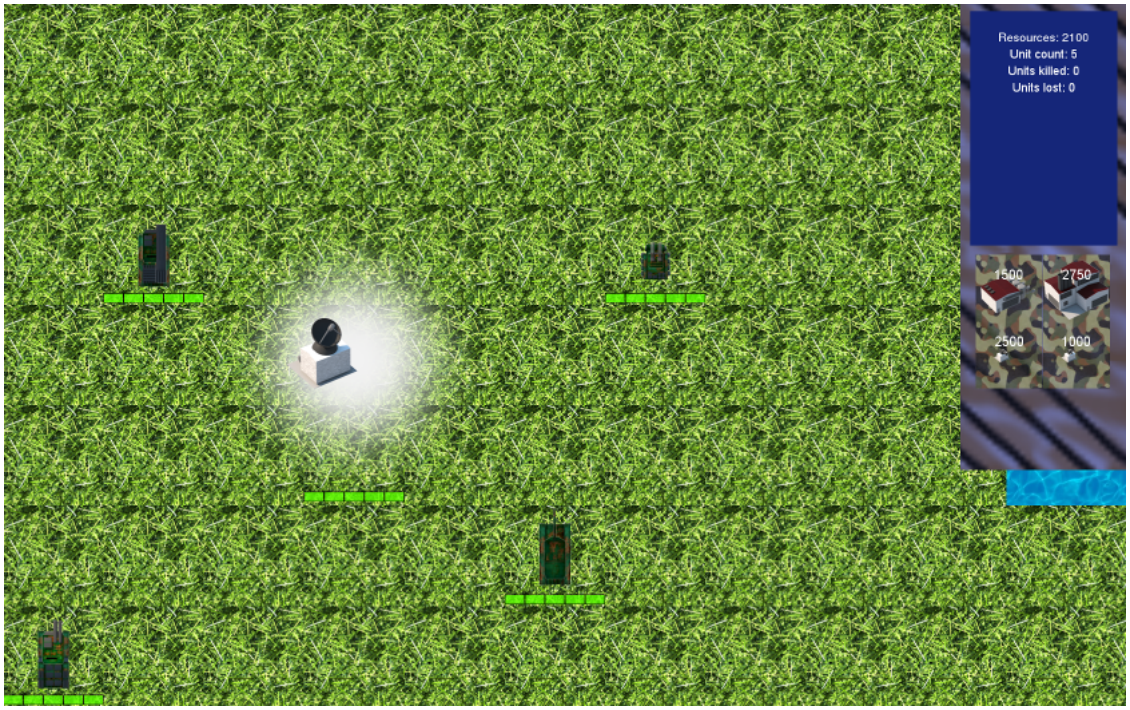
Jádrem technologického dema(mimo framework třídy) je třída *GameMap*, která nese celou herní desku. Vytváří pomocné třídy pro zpracování konfiguračních souborů a také pomocnou logiku hry, která je použita spíše pro uživatele na jednodušší ovládání hry. *GameMap* dále konstruuje instance třídy *Player*, která reprezentuje určitého hráče zapojeného do hry. Jsou mu přiřazeny finanční zdroje, identifikace a také jednotky, které vytváří. Objekt *GameMap* je pouze jeden, avšak nebyl pro něj použit návrhový vzor[2] jedináček z toho důvodu, že není potřeba pro něj vytvářet globální přístup, protože jeho použití je pouze na nejvyšší vrstvě hry, a také v modulu `UI`. Toto je tedy hlavní svazek mezi hrou a modulem, který jsem nebyl schopen v této práci zatím odstranit. Řešením by nejspíše bylo vytvoření takového obecného rozhraní, které by dokázalo zajistit universálnost mezi aplikacemi v různých programovacích jazycích.

Třída *GameMap* tedy vytváří svět podle inicializačního souboru *test_map_1.json*, kde zjistí počet hráčů, jejich zdroje, velikost herní desky, počáteční stav základen hráčů a také terén. Herní deska je repezentována dvourozměrnou maticí objektů třídy *Field*, které značí políčko desky. Tato políčka nutně musí mít zvolený terén(třída *Terrain* – velmi jednoduchá, není nutno detailně popisovat) a mohou nést objekty(entity hry). Každé políčko má označení řádku a sloupce, ve kterém se nachází a vlastní atributy, které značí, že se po něm může nějaká jednotka vypravit nebo dá se na něm aktuálně postavit budova. Názorná ukázka je obrázek 4.1, kde vidíme již zmíněný počáteční stav hráče.

Pro uživatele bylo vytvořeno jednoduché rozhraní podobné strategickým hrám dnešní doby(viz. *Red Alert*). Jedná se o postranní panel značící hlavní informace o uživateli(zdroje atd.) a ikonky s možnými stavebními prvky. Kliknutím na grafické rozhraní tříd *Panel*

[1] Volně šiřitelná knihovna v jazyce C pro práci s JSON jazykem

[2] Šablona softwarového inženýrství usnadňující řešení konkrétních problémů v návrhu počítačových programů.



Obrázek 4.1: Snímek technologického dema na počátku herní partie.

herní framework zaznamená místo kliknutí a přeposílá zprávu do *GameLayer*(tzv. „herní vrstva“), která řeší uživatelské vstupy do hry. Herní vrstva je abstrakcí nad veškerým děním ve hře, pouze zpracovává vstupy uživatele a posílá zprávy do *GameMap*, která startuje různé události(např. označení jednotky a následné vypravení na dané místo).

Mechaniky hry jsou řešeny mezi objekty navzájem. Při vytvoření objektu herní deskou se přiřadí konkrétnímu hráči a objekt se naplní parametry(4.2.2) potřebnými k existenci. Entity hry jsou potomci třídy *GameObject*, která je prvním rodičem specifičtějších tříd. Pro jednotky je to třída *Unit*, pro budovy *Building*. Tyto struktury se ještě dále specifikují až na samotné vozidlo nebo továrnu. Každá z těchto vrstev má určitý význam, například *Unit* řeší pohyb entit, zatímco *Building* žádný pohyb nemá. Střelbu objektů jsem implementoval do tříd *Unit* a *Defense*, které při útoku vytváří daný typ *Projectile*. Projektil je střela a je jí předán odkaz na objekt, na který se má vystřelit. Při dosažení pozice, kde měla střela skončit, zavolá *callback*[3] na cílový objekt, který odečte daný počet zdraví předán jako parametr. Pokud této jednotce klesne zdraví na nulu(nebo níže), volá *callback* do vyšších vrstev hry, kde následně probíhá aktualizace dat hráče a vyčištění objektu i políčka.

Hledání okolních nepřátel zajišťuje také hra. Nepočítá se zde s obrovskými počty jednotek, proto každá jednotka prochází celý seznam nepřátelských jednotek a porovnává vzdálenost s vlastním poloměrem dohledu. Pokud je rozdíl pozic na herní desce menší než tento poloměr, kontroluje to samé na poloměr dosahu zbraní a pokud platí i tato podmínka, jednotka začne na cíl střílet. Toto chování reprezentuje uživatelovi jednotky. `UI` však již při spatření nepřítele na dohled reaguje jistými akcemi, které jsou popsány dále(4.3.2).

[3] Adresa metody(v tomto případě i objektu) nebo funkce předána danému objektu nebo modulu k pozdější invokaci nebo zavolání.

4.2 Aplikační rozhraní modulu

Následuje popis aplikačního rozhraní, kde bude vysvětleno, jak se definují hodnoty parametrů hry, entity a další problémy ve vztahu s rozhraním modulu.

4.2.1 Hodnoty hráče

Nejdříve je nutno zadefinovat, co určuje hráčskou sílu v poli. To znamená jakým vojskem hráč pravděpodobně disponuje. Přestože jde o jistou a přesnou hodnotu, stále jde jen o číslo a není tedy možno určit, který typů entit a jaký počet hráč vlastní, dokud neodhalí všechny nepřátelské jednotky na herní desce. Hodnoty hráče jsem rozdělil na části zvolené tak, aby šlo dobře pochopit, v kterých situacích je nutno tyto hodnoty znát. Jde o následující

- hodnota hráče
- hodnota velitele
- hodnota entity

Podrobnější popis těchto částí následuje dále. Bude na ně odkazováno v dalších částech tohoto dokumentu.

Hodnota hráče

Hráč na počátku disponuje základním kapitálem měny hry, počtem jednotek a budov na definovaných místech(definice v konfiguračním souboru). Ačkoliv v průběhu hry nabývá nových sil a zdrojů, může tyto faktory také ztrácet. Proto má každý hráč tzv. „hodnotu“, která je vyhodnocena složitou funkcí obsahující sjednocení parametrů všech jednotek a budov i mírou financí. Tato hodnota se mění dynamicky za běhu hry a je obsažena v rozhodování pro danou strategii umělého protivníka.

Hodnota velitele

Při vytváření instancí třídy *Commander* dochází k určení hodnoty armády, která se tomuto veliteli přiřazuje. Při ztrátě jedné z přiřazených jednotek se hodnota přepočítá. Tímto velitel určuje podle map ohrožení nebo podle aktuálních nepřátelských jednotek v dosahu reakci, kterou je nutno provést v rámci splnění cílů hry(např. zaútočit nebo prchat).

Hodnota entity

Hodnota určující sílu entity dle definovaných parametrů(podrobnější popis náseduje hned v další podsekci Aplikačního rozhraní(4.2) – Definice entit(4.2.2)).

4.2.2 Definice entit

Entity hry obsahují parametry potřebné k jejich analýze a vyhodnocení. Parametry definují entitu a její chování je určeno stavy entity. Vždy se musí nacházet právě v jednom z těchto stavů – určují chování a napomáhají ve vyhodnocení společně s jejími parametry. Stav entity je tedy určen jejími parametry a danou herní situací, ve které se nachází. Parametry entit se dělí na hlavní (4.2.2) a vedlejší (4.2.2). Hlavní parametry jsou nutné k existenci entity a její hodnoty rovněž určují její význam. Vedlejší parametry napomáhají v rozhodování a mohou

také ovlivnit stav entity. V rámci tohoto modulu jsem navrhnul další rozdělení parametrů vzhledem k reálnému vyhodnocování hodnot entit(4.2.2).

Hlavní parametry

- Název – alfanumerická identifikace entity, musí být unikátní a její využití je spíše v samotné hře, než v modulu
- Zdraví – celočíselná hodnota, která určuje výdrž entity a při poklesu na nulu entita končí svou životnost
- Poškození – poškození, které je entita schopna vydat v jednom výstřelu(za výstřel považujme jakýkoliv útok směrem k cíli) a poškozuje zdraví cíle, je celočíselná
- Rychlost pohybu – doba, za kterou objekt urazí jednotku vzdálenosti ve hře a její hodnota je v plovoucí řádové čárce
- Rychlost střelby – čas, který je nutný čekat mezi jednotlivými výstřely entity(tzv. doba nabíjení)

Vedlejší parametry

- Typ entity – alfanumerické určení, po jakém terénu je jednotka schopna se pohybovat
- Doba vytvoření – čas nutný k vytvoření jednotky na stanovené pozici
- Poloměr dohledu – vzdálenost, na kterou je jednotka schopna „vidět“, odhalit nepřátelské entity
- Poloměr dosahu zbraní – dosah útoku jednotky, maximální vzdálenost, na kterou je entita schopna střílet
- Konstrukční poloměr – určuje na jakou vzdálenost od této entity je možno postavit jinou budovu nebo jednotku
- Cena – celočíselná hodnota, která určuje jaké množství zdrojů je nutno k vytvoření jednotky
- Typ projektilu – tento parametr je v technologickém demu pouze pro grafické znázornění útoku, ale v jiných hrách může určovat i to, co má po útoku nastat za událost(parametr je velmi obecný, jeho implementace je nutno specifikovat v dané hře, např. že útok zasáhne více cílů)

Další rozdělení parametrů

Vzhledem k určení, které parametry jsou relevantní hodnotě entity(4.2.1), bylo potřeba rozdělit parametry na bitevní a produkční. Bitevní parametry jsou relativní vzhledem k bojovým schopnostem entity, zatímco produkční parametry ovlivňují pouze plánování a ekonomickou situaci hráče, nikoliv dílčí střety, ale mají také vliv na celkový průběh hry stejně jako ty bitevní.

- Bitevní parametry

  - zdraví
  - poškození
  - rychlost pohybu
  - rychlost střelby
  - poloměr dohledu
  - poloměr dosahu zbraní
  - typ entity
- Produkční parametry
  - doba vytvoření
  - cena
  - konstrukční poloměr
- Ostatní
  - název
  - typ projektilu

Příklad z konfiguračního souboru

Zde se nalézá příklad dvou entit použitých v technologickém demu. Jde o srovnání nejslabší a nejsilnější jednotky ve hře(výzvědný tank(4.2.2) a raketový tank(4.2.2)). Každá má své silnější i slabší stránky. Tyto faktory mohou významně ovlivnit rozhodování `UI`, pokud by implementace dosáhla potřebné složitosti(zohlednění veškerých faktorů a reakce na veškeré podněty). Jak již bylo zmíněno, soubor je psán v jazyce JSON a obsah je v anglickém jazyce(z důvodu generalizace zdrojových kódů). [11].

```
{
"name":"scout_tank",
"health":60,
"damage":9,
"type":"ground",
"movement_speed":0.7,
"fire_rate":1.3,
"build_time":10,
"cost":450,
"view_radius":6,
"range_radius":3,
"projectile":"shell",
"build_radius":0
}
```

Obrázek 4.2: Konfigurace výzvědného tanku

```
{
"name":"missile_tank",
"health":75,
"damage":60,
"type":"ground",
"movement_speed":1.7,
"fire_rate":3.0,
"build_time":45,
"cost":1750,
"view_radius":6,
"range_radius":6,
"projectile":"rocket",
"build_radius":0
}
```

Obrázek 4.3: Konfigurace raketového tanku

Na levé straně se nachází výzvědný tank(4.2.2), je to nezákladnější jednotka ve hře a také nejslabší. Její přednosti jsou rychlost pohybu a poloměr dohledu, cena a doba vytvoření. Přednosti každé jednotky určují její podstatu ve hře, její účel, kterým je prospěšná každému hráči. Zatímco výzvědný tank je rychlý, ztrácí však na zdraví a poškození, jeho opakem je

raketový tank na pravé straně(4.2.2), který disponuje obrovskou palebnou silou a dosahem zbraní. Ztrácí však na zdraví, době vytvoření a ceně.

Z těchto faktů můžeme odvodit závěr této podsekce. I pokud by název výzvědného tanku nebyl takový jaký je, z analýzy všech jednotek by vyplynulo jeho opodstatnění. Slouží tedy k výzvědným misím a je klíčový v plánování velkých soustředěných útoků na již prozkoumaná místa. `UI` jej tedy bude používat k nalezení lokace hráče a zjištění jeho alespoň částečné hodnoty(4.2.1). Raketový tank je naopak tzv. palebná podpora. Jeho účel je držet se zpátky a střílet ze zálohy, protože je zranitelný. Bylo by tedy velmi nevhodné jej držet v první linii, stejně tak jako nechat jej bez obrany.

Opravdovou výzvou pro vývojáře strategických her je tedy přimět umělého protivníka „přemýšlet“ nad podstatou každé jednotky. V mnohých hrách jsou však role každé jednotky přímo definovány a je také definováno jak s nimi zacházet. Parametry jednotek se mohou také měnit s určitým postupem „evoluce“ ve hře nebo vylepšeními. Bylo by potřeba analyzovat každý parametr a porovnávat s ostatními, ohodnotit jej podle nějaké heuristiky. V této práci jsem však nebyl schopen toto chování implementovat a zároveň tato problematiky přesahuje rámec bakalářské práce.

4.3 Jádro modulu

Pro popis tříd, který bude nyní následovat, informaci o nich můžete získat v podkapitole architektura(3.4).

Hlavní třídou modulu je *Interface*, jejíž instanci drží hra, která běží na jednom vláknu procesoru. Modul vyžaduje, aby mu hra věnovala tok programu vždy po nějakém intervalu. Tento interval byl nastaven na tři sekundy pro hlavní výkonnou část a tím je třída *Computer*. Tuto třídu vytváří rozhraní a reprezentuje jednoho počítačového protivníka. V technologickém demu je však pro jednoduchost použit právě jeden hráč a jeden umělý protivník.

4.3.1 Strategie počítače

Computer při vytvoření instance dále vyžaduje další objekty, a to přístup k herní desce hry, která je reprezentována šachovnicí(2D matice), přístup k parametrům entit(analýza souboru) a instanci třídy *StrategyController* reprezentující imperativní chování `UI`. Tato definice se nalézá v souboru *computer_default.json* a říká, čeho musí počítač dosáhnout. Jde o vrstvy různých cílů, které udávají postup od základních objektů hry k těm pokročilým. Toto chování by se dalo nazvat jako „přímá inteligence“, kdy jsou jasná pravidla co musí protivník provést. Opačný způsob je však velmi náročný a vyžaduje kvalitní analýzu herních parametrů tak, aby byl počítač schopen sám rozhodnout, které budovy slouží k čemu, a jaké jednotky využít za daným účelem. Pokud by se mi naskytla příležitost pokračovat v této práci, a to v rámci diplomové práce, velmi rád bych na tuto možnost přistoupil.

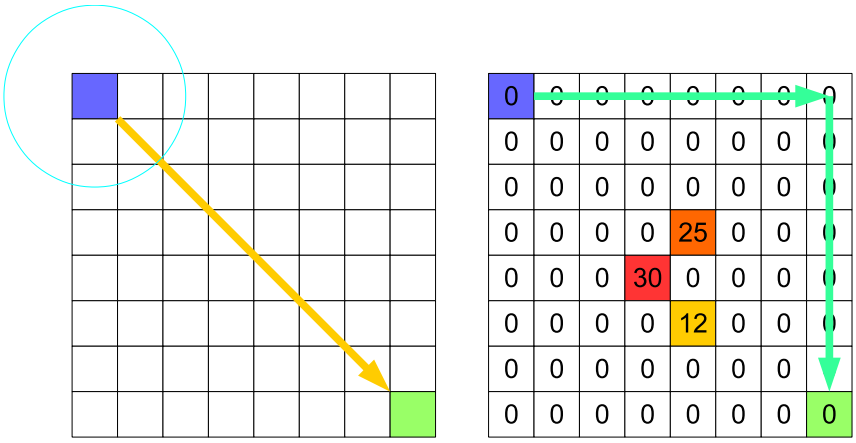
Přímé cíle počítače jsou rozděleny na potřebné a občasné(v souboru se vyskytují řetězce *MANDATORY* a *OCCASIONAL*). Potřebné cíle jsou takové akce, které vedou k růstu armády. Hlavní cíl zničení nepřátelské základny je implicitní, počítač se snaží redukovat hodnotu protihráče(4.2.1) na minimum. Občasné cíle jsou například výzvědné mise, které vedou k aktualizaci mapy ohrožení(3.5.2) a získání informací o tom, s jakou silou nepřítel disponuje. Mezi občasné cíle patří výzvědná mise a útok. Počítač volí jednotky pro útok sám, a to podle své hodnoty(4.2.1) vůči těm, které na výzvědné misi zjistil.

4.3.2 Plánování

Plánovací třída *Computer* vždy při přiděleném toku programu kontroluje stanovené plány. Při dosažení podmínek pro daný cíl ihned instanciuje třídu *Commander*(česky velitel), která slouží pro lokální řešení pohybů jednotek. Přidělí se odkazy na určené objekty, kterými bude velitel disponovat a také výčtovou hodnotu značící typ cíle(v programu se vyskytují hodnoty *SCOUTING* a *SND*[4]). Ihned po vytvoření se spočítá jeho hodnota(4.2.1), jejíž význam určuje chování vojsk na bojišti. Dále budou popsány dva typy misí, které počítač uskutečňuje pomocí velitele.

Výzvědná mise

`UI` volí jednotky s nejvýšší rychlostí pohybu k tomu, aby se volně pohybovali po herní desce a nalézali nepřátelské entity. Těmito entitami mohou být vozidla nebo budovy. Při spatření vozidel si velitel zjistí jejich hodnoty(4.2.1), a pokud jsou v součtu nižší než celková hodnota velitele, pokusí se zastavit hrozbu ihned nebo uniknout. Jelikož jde o výzvědnou misi, nepředpokládá se, že počítač zvolí mnoho sil. Vybere totiž náhodný počet těch nejrychlejších, které jsou k dispozici(nejsou přiděleny žádnému veliteli). Toto číslo je v rozmezí intervalu $< 2; 4 >$ a také se nachází v konfiguraci mise.



Obrázek 4.4: Vizualizace hledání cest s pomocí mapy ohrožení

Pokud velitel ztratí dohled určité nepřátelské jednotky, uloží její hodnotu do mapy ohrožení(Obr. 4.4) a budovy naopak do mapy cílů(Obr. 4.5). Zde může ovšem teoreticky docházet k omylům, protože mapuje pouze hodnoty nepřátelských entit. To znamená, že pokud spatřil nějaké vozidlo vícekrát, bude v mapě více záznamů a tím také mylná představa o protihráči. Předejít tomuto problému by se dalo jednoduše ukládání unikátní identifikace objektu spolu s jeho hodnotou. Vezměme si však příklad z lidského uvažování.

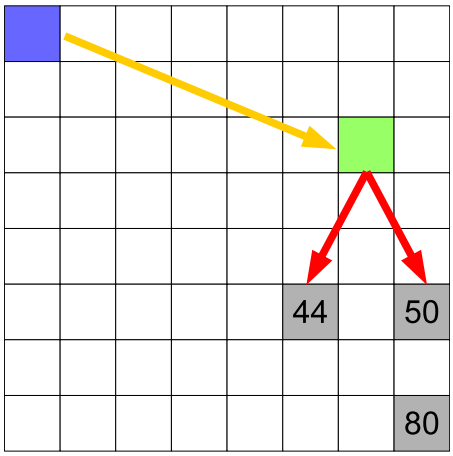
Člověk nerozezná danou jednotku od té druhé(pokud jde o stejný typ) nevidí–li nějaké specifičtější informace o ní. V technologickém demu má však hráč možnost vidět pouze zelené čtverečky značící zdraví entity. V případě, že spatří stejný typ jednotky vícekrát na různých místech v poli, nemusí vědět, že jde pořád o tu samou. Tímto bych chtěl naznačit fakt, že by ukládání identifikace do mapy mohla být jistá forma podvádění vůči hráči. Pokud má být hra spravedlivá a počítač se má chovat „lidsky“, nelze tuto možnost připustit.

[4]Najdi a znič(angl. *Search and Destroy*) je vojenská taktika využívaná například americkými vojsky během války ve Vietnamu

Člověk uvažuje velmi podobně ve formě těchto datových struktur(2.2.2), proto jsem je implementoval.

Najdi a znič

Aktualizace datových map během výzvědné mise se zde provádí taktéž, ovšem za jiným cílem, než jen prozkoumat herní pole. Cílem je zde vyhledat na mapě cílů, kde se nachází velké množství budov a zvolit ke shromáždění místo poblíž dostatečně daleko tak, aby nedošlo k prozrazení plánu a zaútočit na danou lokaci s cílem vyhrát herní partii. Sílu vojska velitel zvolí součtem všech hodnot v mapě ohrožení. V případě, že je mapa prázdná, naordinuje k útoku vše, co má. Zároveň pokud je prázdná i mapa cílů(zvědové dosud neodhalili nepřátelskou základnu), v tomto případě nemůže zaútočit. Musí počkat, dokud se alespoň jedna z těchto map nenaplní konkrétnějšími daty. Ve skutečnosti by armáda určitě neposílala hlavní údernou sílu na místa, o kterých jí chybí veškeré informace.



Obrázek 4.5: Vizualiace hledání místa shromáždění při útoku

Počítač bude hledat vhodné místo ke shromáždění tím způsobem, že nahlédne do mapy ohrožení i mapy cílů(Obr. 4.5) tak, že prvně zjistí kde se nachází nejvíce koncentrovaný shluk nepřátelských budov. Místo zvolí na vzdálenost dvakrát větší, než je největší možný dohled jedné z přiřazených jednotek a pomocí mapy ohrožení volí cestu. V případě, že se v okolí místa setkání vyskytne nepřítel, ihned zautočí na danou hrozbu. Při dosáhnutí místa všemi jednotkami velitel okamžitě táhne do boje a přikazuje útoky jednotkám na ty, které stojí v cestě za cílem.

4.3.3 Analýza základny

Při startu hry se hráč ocitá na určitém místě s hlavní budovou(3.3.1) a malým počtem jednotek kolem ní. Jak již bylo zmíněno v návrhu(3), musí zjistit jak situovat svoje budovy. Na obrázku 4.6 je část příkladné základny, kde hlavní budova je zelený čtverec a červená šipka znamená předpokládaný útok nepřítele. Šedé čtverce znamenají překážku, přes kterou nemohou jednotky projít.

Člověk logicky soustředí obranu na místa, kudy přichází nepřátelé a továrny umístí někam, kde budou z dosahu případných útoků. Toto chování jsem implementoval i umělému protivníkovi. Formu tohoto uvažování zapouzdřuje třída *BuildMap*, která je vytvářena ihned

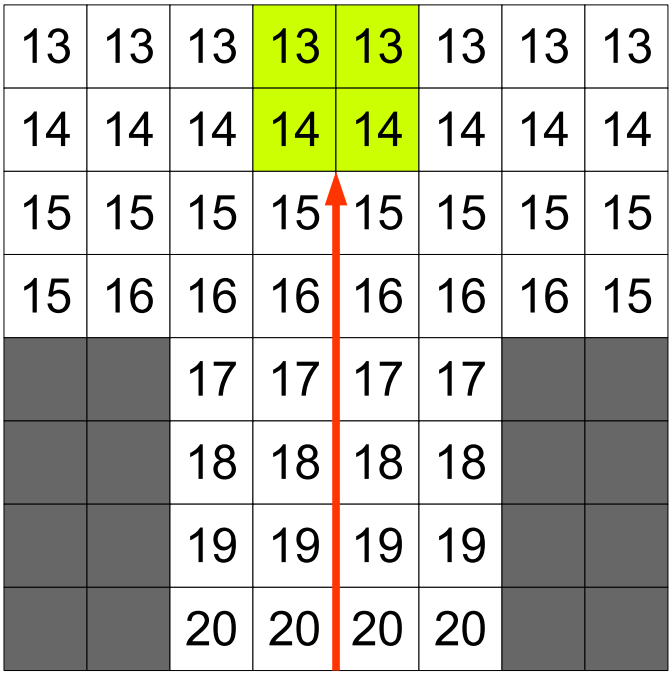
po instanciaci *Computer*. *BuildMap* využívá mnou navržený algoritmus hledání příchodových cest do základny(1). Algoritmus je vázán na několik parametrů hry, nicméně nebyl navržen zatím dost obecně na to, aby mohl být použit v jakékoliv hře. Třída obsahuje metody k výpočtu a také matici obsahující hodnoty, které značí kam přibližně umístit dané budovy. Uvnitř matice se nachází pouze celá čísla značící vyššími čísly umístění obranných věží a nižšími(blízko k nule) továren. Algoritmus využívá také jednu z informovaných metod prohledávání stavového prostoru konkrétně A*(2.3.2), která slouží k nalezení cest od nepřátelské základny k té vlastní.

Algoritmus analýzy okolí základny

Následuje výčet a definice dat, které algoritmus potřebuje k výpočtu:

- Lokace protihráčovy základny – **LPZ**
- Konstrukční poloměr(4.2.2) – **KP**
- Matice herní desky – **MHD**
- Lokace vlastní základny – **LVZ**

Výchozí bod je lokace vlastní základny a výstupem algoritmu bude matice o velikosti $2*KP^2$ s názvem **VM**. Dále následuje algoritmus pro ohodnocení buněk výstupní matice(2) z výpočtu okolí základny. Jeho vstupem je tedy nulová matice a pole bodů, které reprezentuje vchody do základny(proměnná názvu **VZ**) a výstupem je matice upravená.



Obrázek 4.6: Vizualiace analýzy základny

Algoritmus 1: Algoritmus analýzy okolí základny

Input: $\{LPZ, KP, MHD, LVZ\}$
Output: VM

1: Od výchozího bodu procházej buňky MHD v každém směru(konkrétně vlevo, vpravo, nahoru a dolů), dokud počet prošlých buněk nedosáhne KP nebo nenarazíš na překážku a nalezené body ulož.
2: Mezi body v protichůdném směru vytvoř pole buněk, které reprezentuje přímku a tyto body ulož.
3: Projdi oba kolmé směry na tyto přímky tak, jako v bodě 1 a ulož každý konečný bod vycházející z právě jednoho bodu v přímce.
4: Ulož počáteční a konečné nalezené body hledaných z přímek, mezi zbylými body porovnávej jejich polohu a pokud se liší jejich kolmá vzdálenost(x na y nebo y na x), oba lišící se body ulož.
5: Nalezené body posouvej zpět a kontroluj jejich kolmé buňky, pokud je zde překážka, zastav posouvání bodu, v opačném případě při dosažení výchozí pozice, vrať bod zpět.
6: Z těchto bodů generuj linky na oba kolmé směry a pokud dojde k protnutí s jiným bodem, výsledek protnutí ulož.
7: Vygeneruj pole buněk takové, které obsahují vždy dva zmíněné rozdílové body, bod protnutí a zbylé body na cestě mezi nimi.
8: Metodou **A*** jdi od LPZ směrem k LVZ a nalezni tímto všechny možné cesty.
9: Poslední možné setkání s vygenerovaným polem výsledných buněk označ jako validní, jde o nalezené průchody.
10: Vygeneruj VM tak, že ohodnotíš výsledná pole algoritmem ohodnocení výstupní matice(2).

Algoritmus 2: Algoritmus ohodnocení výstupní matice

Input: $\{VM1, VZ\}$
Output: $VM2$

1: Z každého bodu jdi do všech osmi směrů a hodnotu bodu dekrementuj.
2: Pokud narazíš na již ohodnocenou buňku a hodnota je alespoň stejná, dále nepokračuj a vyber jiný směr.
3: Pokud narazíš na hranici matice, vyber jiný směr.
4: Po vyčerpání směrů a výchozích bodů skonči.

4.4 Implementace výpočtů herních hodnot

Třída *EvalParser* zpracovává konfigurační soubor, který obahuje ohodnocení každého parametru entity ve hře. Je dáno, že tyto parametry jsou povinné v každé možné hře, kterou by modul mohl doprovázet. Nicméně parametry mnou definované jsou z vlastní zkušenosti velmi časté v jiných hrách, proto jsem takové zvolil. Uvedeme si zde opět příklad dvou parametrů z tohoto souboru:

```
{
"name":"damage",
"value":10,
"type":"ascending",
"ascend_value":1
}
```

Obrázek 4.7: Konfigurace parametru poškození

```
{
"name":"fire_rate",
"value":5,
"type":"descending",
"ascend_value":0.1
}
```

Obrázek 4.8: Konfigurace parametru rychlost střelby

Vysvětlivky:

- "name"– název parametru **n**
- "value"– hodnota jedné jednotky parametru **v**
- "type"– vzrůstající nebo klesající **t**
- "ascend_value"– jednotka hodnoty **a**

Každý parametr má danou jednotku, pro kterou je určená hodnota této jednotky. Na levém obrázku(Obr. 4.1) si můžeme všimnout, že jednotka poškození je jedna, její hodnota je deset. Její vlastnost je vzrůstající(angl. *ascending*), to znamená, že při zvýšení počtu jednotek poškození se zvedá hodnota parametru. Výpočet hodnot vzestupných probíhá jednoduše vzorcem 4.1 a sestupných zase vzorcem 4.2.

$$h = \frac{v}{a} \cdot p \tag{4.1}$$

$$h = \frac{\frac{1}{p}}{\frac{a}{v}} \tag{4.2}$$

Výpočet hodnoty entity(4.2.1) tedy probíhá následově. Je potřeba si uvědomit, co jednotlivé parametry znamenají a jaký mají přínos pro jednotku. Velmi častým faktorem entit v jiných strategických hrách je hodnota „poškození za sekundu"(angl. *DPS - damage per second*). Uvádá, kolik poškození je objekt schopen vydat za jednu sekundu. Proto budeme hodnoty poškození a rychlosti střelby násobit(opak jejich jednotek – $\frac{poškození}{rychlost}$). Tímto jednotka s rychlejší střelbou, ale menším poškozením než ta jiná, může mít stále vyšší hodnotu. Vzorec pro výpočet všech bitevních parametrů(4.2.2) je vzorec 4.3, kde dané proměnné znamenají:

- v – hodnota entity
- h – zdraví
- d – poškození

- f – rychlost střelby
- m – rychlost pohybu
- w – poloměr dohledu
- r – poloměr dosahu zbraní

$$v = h + (d \cdot f) + m + w + r \tag{4.3}$$

Z příkladu jednotek(4.2.2) vezměme hodnotu poškození výzvědného tanku. Jeho poškození je $p = 9$. Výpočet hodnoty h tohoto parametru proběhne vzorcem pro výpočet vzestupné hodnoty 4.4.

$$h = \frac{v}{a} \cdot p = \frac{10}{1} \cdot 9 = 90 \tag{4.4}$$

Na pravém obrázku(Obr. 4.2) vidíme ohodnocení rychlosti střelby. Výzvědný tank má rychlost střelby $r = 1,3$. Hodnota je klesající, to znamená, že pokles jednotek hodnoty má vzrůstající následek hodnoty parametru(čím rychlejší střelba, tím více poškození). Výpočet hodnoty rychlosti střelby je potom vzorec 4.5 a

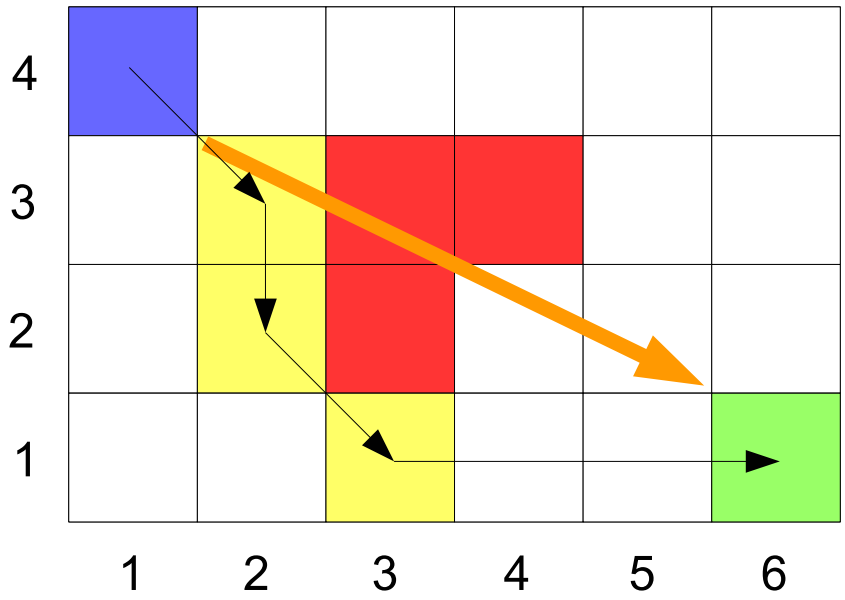
$$h = \frac{\frac{1}{p}}{\frac{a}{v}} = \frac{\frac{1}{1,3}}{\frac{0,1}{5}} = 38,461 \tag{4.5}$$

Nyní známe hodnotu výzvědného tanku t_1 po výpočtu dvou z jeho parametrů a hodnota je $t_{1_{v_2}} = d_1 \cdot f_1 = 3461,49$. Přeskočíme nyní ty samé výpočty pro raketový tank t_2 z příkladu(4.2.2) a vezměme rovnou jeho výsledek $t_{2_{v_2}} = d_2 \cdot f_2 = 10000$. Zde vidíme, že druhý tank je v palebné síle skoro třikrát účinnější. Z tohoto částečně plyne podstata obou vozidel, kterou jsem zmínil v podkapitole návrhu(4.2.2). Tyto výpočty vznikly na základě vlastního uvažování o vyváženosti hry.

4.5 Nalezení cest k cíli

Hra potřebuje na úrovni jednotek rozlišit příkazy k přesunu mezi hráčem a počítačem. `UI` si pro každou jednotku drží frontu traťových bodů(pozice, kterými musí jednotka projít), které byly nalezeny na cestě k cíli. Počet bodů záleží na počtu překážek nerovnoměrně.

Na obrázku (Obr. 4.9) vidíme jednoduché znázornění situace, kdy se z modrého políčka potřebuje jednotka dostat do zeleného(přímé znázornění oranžovou šipkou). V cestě stojí nějaká překážka znázorněná červeně. Metodou **A***(2.3.2) modul nalezne nejkratší cestu a vytvoří traťové body znázorněné žlutou. Tyto body je potřeba umístit tak, aby se jednotka vždy pohybovala přímo na bod to znamená, aby nedošlo k přílišnému komplikování samotného algoritmu posunu a entita se nějakým způsobem nezablokovala. Černé šipečky jsou už jednotlivé příkazy k pohybu uložené ve zmiňované frontě pro danou jednotky. Tyto fronty budou uloženy do asociativního pole[5] s klíčem jednoznačné identifikace objektu.



Obrázek 4.9: Vizualiace hledání cest a vytváření traťových bodů

4.6 Testování inteligence

Proběhly dva typy testů. Prvním testem bylo testování algoritmu na rozvoj základny počítače, kdy předmětem sledování byla výstupní matice a samotné akce protivníka. Druhým testem bylo zkoumání plánování počítače při útoku na základnu hráče, kde sledované byly struktury map cílů a ohrožení, také označení míst ke shromáždění před daným útokem.

4.6.1 Testování rozvoje základny počítače

Vstupem testu je inicializační soubor základny počítače(*test_map_1.json*), kde je dána pozice hlavní budovy(*headquarters* hráče č. 1). Výstupem testu je matice nesoucí hodnoty míst, kde má počítač postavit obranné nebo produkční budovy.

[5]Pole typu klíč – hodnota. Přístup na hodnotu pomocí klíče.

Při umístění produkční budovy protivník prochází tuto matici od nejnižší hodnoty a zkouší ji umístit. Při neúspěchu pokračuje dále, dokud nenarazí na vhodné místo. V opačném případě hledá nejvyšší hodnotu a to k postavení obranných věží. Je zde možnost matici vidět přímo v technologickém demu, a to zbarvením jednotlivých políček na místě protihráčovy základny, kdy světlejší políčka značí umístění obranných věží, zatímco tmavší jsou pro produkční budovy. Výstupy testů jsou dostupné ve složce *samples/* v příloze CD.

4.6.2 Testování postupu vojsk počítače

Vstupem testu je spuštěná herní partie a výzvědný tým umělého protivníka. Výstupem je aktualizace mapy cílů s lokací nepřátelských budov a jejich hodnotou, mapy ohrožení s místy, kde se nacházely nepřátelské jednotky a jejich hodnoty.

K těmto datům je potřeba, aby počítač vyslal výzvědný tým, který skoro náhodně krouží po herní desce a hledá nepřátele. Při plánování útoku počítač hledá místo ke shromáždění vojska podle mapy cílů na vzdálenost dvakrát větší, než je nejvyšší dohled jedné z jednotek. Velikost vojska volí dle získaných informací v mapě ohrožení a také se snaží těmto místům vyhnout, pokud je jeho vojsko v nižší palebné síle než zaznamenáno v této struktuře. V technologickém demu se místa shromáždění zabarvují na červeno přímo v herní desce hry. Výstupy testů jsou dostupné ve složce *samples/* v příloze CD.

Kapitola 5

Závěr

Podařilo se navrhnout a implementovat chování umělých hráčů tak, že se jistým způsobem podobá tomu lidskému při hraní počítačových her. Ačkoliv existuje více různých způsobů jak tento problém zkonstruovat, všechny se určitým způsobem kopírují tak, jak se hry kopírují samy od sebe. Pokračováním by rozhodně bylo aplikací analýzy elementů hry(objektů, jednotek atd.) skrze nějaké rozhraní a vytvořit si vlastní rozhodovací strukturu pro dané situace hry. Počítač by sám „věděl“ co, kdy a jak použít.

Dalším možným příspěvkem pro pokračování by byla konstrukce jednoduchého algoritmu pro učení z předchozích her. Tímto by se umělý protivník stal adaptivním vůči lidskému oponentovi a postupem času reagoval „inteligentněji“ na jeho podněty. Možnost aplikace více způsobů jak „učit“ protivníka(2.5.1) a porovnat tyto metody.

V poslední řadě mně napadlo, jako další příspěvěk aplikovat různé plány pro umělého protivníka jak vyhrát danou hru. Tyto plány(nebo–li strategie) míti rozlišené úrovněmi a testovat lidské řešení proti těmto návrhům. Rozdělit testy dle daných úrovní a sledovat, jak bude člověk uvažovat v daných situacích. Tímto by se dal iterativně zkoumat vývoj strategií člověka a porovnávat jej vývojem strategií počítače. Projekt pro sledování lidských stylů hry by mohl kooperovat s projektem na zkoumání vývoje umělých protivníků.

Tak jako se vyvíjí naše civilizace, tak se také vyvíjí počítačové hry, které jdou kupředu rychleji směrem grafického vzhledu než–li umělou inteligencí. Tato práce rozvinula myšlenku uvažování počítače při bojových situacích ve hrách a implementovala vyspělý přístup k ní. A protože pojem virtuální realita se stává čím dál tím více skutečností, je potřeba k tomu připravit kvalitní základ tak, aby se nám subjekty v ní nejevily příliš „hloupě“.

Literatura

[1] A* search algorithm. online.
URL http://en.wikipedia.org/wiki/A*_search_algorithm

[2] Berg, H. A.: *The Computer Game Industry*. Diplomová práce, Norwegian University of Science and Technology, Department of Telematics, 2010.

[3] Berg, S. S.: *Artificial Intelligence Techniques in Real-Time Strategy Games - Architecture and Combat Behavior*. Diplomová práce, Institutt for datateknikk og informasjonsvitenskap, 2006.

[4] Champandard, A. J.: The Mechanics of Influence Mapping: Representation, Algorithm & Parameters. online, 5 2011.
URL http://aigamedev.com/open/tutorial/influence-map-mechanics/

[5] Gaudl, S.; Davies, S.; Bryson, J. J.: Behaviour Oriented Design for Real-Time-Strategy Games. Technická zpráva, University of Bath, Department of Computer Science, 2013.

[6] Hagelbäck, J.; Johansson, S. J.: Using Multi-agent Potential Fields in Real-time Strategy Games. Technická zpráva, Blekinge Institute of Technology, Ronneby, Sweden, 5 2008.

[7] Ishida, T.; Korf, R. E.: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 17. IEEE Computer Society Washington, DC, USA, 6 1995, 619 s., iSSN: 0162-8828.

[8] Kehoe, D.: Designing Artificial Intelligence for Games. online, 6 2009, poslední aktualizace 1.1.2015.
URL http://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1/

[9] Laird, J. E.; van Lent, M.: Machine Learning for Computer Games. Technická zpráva, University of Michigan, USA, 3 2005.

[10] Shani, G.; Amato, C. (editoři): *High-level reinforcement learning in strategy games*, ročník 1, International Foundation for Autonomous Agents and Multiagent Systems, Richland, 5 2010, iSBN: 978-0-9826571-1-9.

[11] Wong, K. W.: Combining AI Methods for Learning Bots in a Real-Time Strategy Game. online, 10 2008, poslední aktualizace 2009.
URL http://www.hindawi.com/journals/ijcgt/2009/129075/

[12] Zbořil, F.; Zbořil, F.: *Základy umělé inteligence*. 5, VUT v Brně, Fakulta Informačních technologií, 2012, 145 s.

[13] Zeng, W.; Church, R. L.: *International Journal of Geographical Information Science*, ročník 23. Taylor & Francis, Inc. Bristol, PA, USA, 2009, 543 s., iSSN: 1365-8816 EISSN: 1365-8824.

Příloha A

Obsah CD

Adresářová struktura CD:

- doc – složka s textovými dokumenty
- src – složka se zdrojovými kódy
- Resources – zdroje potřebné ke spuštění aplikace

V adresáří *doc* existují další podsložky:

- navod – uživatelský návod na instalaci a příručka k technologickému demu
- projekt – práce ve formátu PDF
- latex – zdrojové kódy a soubory pro textovou verzi práce